Administrators Guide to GARA

March 2000*

* Please check http://www.mcs.anl.gov/qos/ for the latest version of this document.

Contents

Note: Before you read about administering GARA, you should have at least a passing familiarity with Globus. You can learn more about Globus at http://www.globus.org. This guide concentrates on describing GARA for someone that needs to install and maintain GARA. If you need information on programming GARA, please see the Programmers Guide to GARA. *If you would like more information about the research being done with GARA, please see the papers available at the Globus web site.*

About GARA

The GARA architecture provides programmers with convenient access to end-to-end quality of service (QoS) for programs. To do so, it provides mechanisms for making QoS reservations for different types of resources, including computers, networks, and disks. A reservation is a promise from GARA that an application will receive a certain level of service from a resource. For example, a reservation may promise a certain bandwidth on a network or a certain percentage of a CPU.

The GARA architecture is defined as a layered architecture with three levels of APIs and one level of low-level mechanisms:

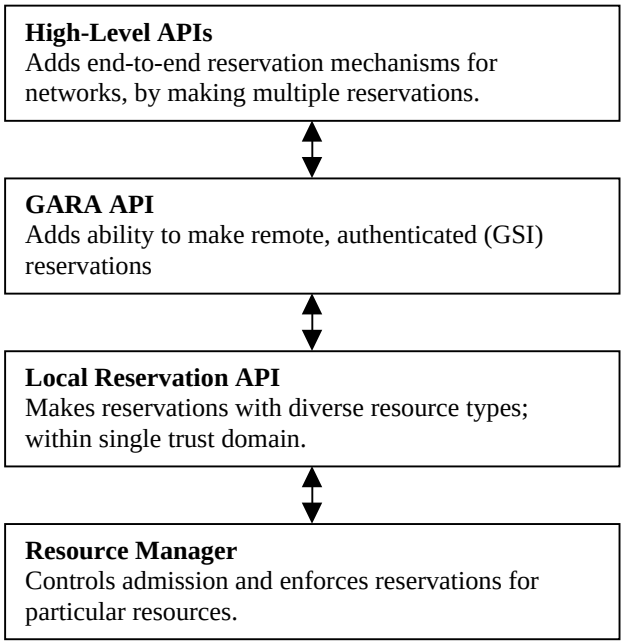


Figure 1 – GARAÕs layered architecture

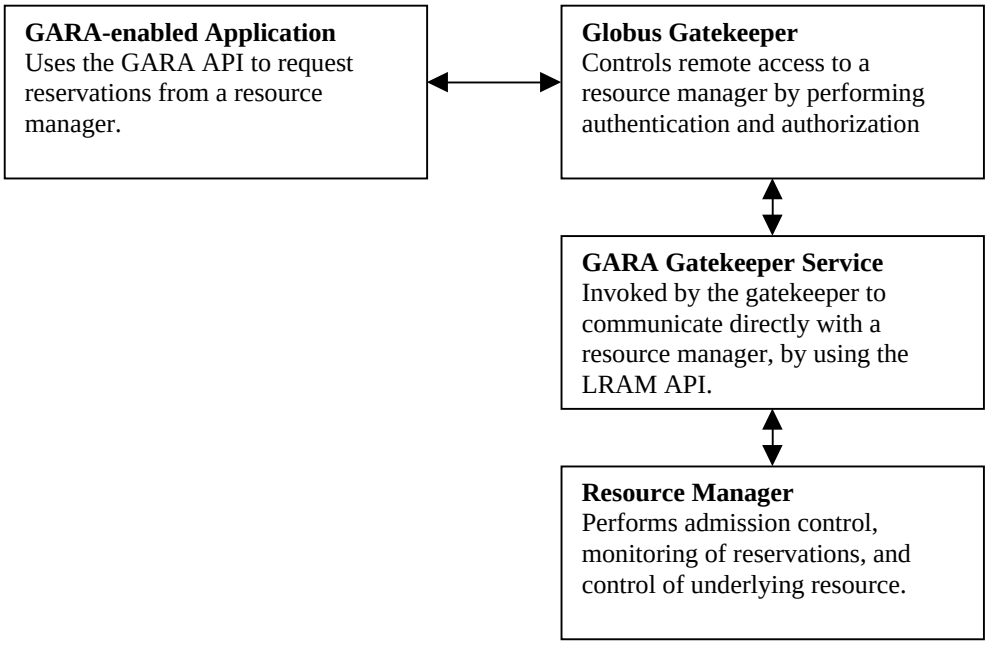
Note that ÒGARAÓ refers to two things: the ÒGARA Architecture,Ó which refers to the entire diagram above, and the ÒGARA API,Ó which is the API for making a single reservation.

The GARA API has two interesting advantages. First, it allows you to make reservations either in advance of when you need them or right at the time that you need themÑan *immediate*

reservation. Second, you use the same API to make and manipulate a reservation regardless of the type of the underlying resource, thereby simplifying your programming when you need to work with multiple kinds of resources.

The GARA API can be considered a remote procedure call mechanism to communication with a resource manager. A resource manager controls reservations for a resource: it performs admission control and controls the resource to enforce the reservations. Some resources already have the ability to work with advanced reservations, so the resource manager is a simple program. Most resources cannot deal with advanced reservations, so the resource manager tracks the reservations and does admission control for new reservation requests. Much of the research in GARA has focused on building useful resource managers.

It is important to understand how this remote procedure call works:



When a program uses the GARA API to communicate with a resource manager, the communication does not happen directly, but happens through the assistance of the Globus gatekeeper. The gatekeeper performs three important service: authentication, to verify the identity of the person making the reservation, authorization, to verify that the person is allowed to make a reservation, and finally, the launches the gatekeeper service to handle the communication with the resource manager.

When you are installing and configuring GARA, you will need to adjust all of these portions:

- You will need to compile the gatekeeper service and resource manager.
- You will need to configure the gatekeeper with the location of the GARA gatekeeper service so that it can launch the service in response to requests from applications.
- You will need to configure the resource manager so that it understands the properties of the underlying resource that it is controlling.

Resource Managers

Currently, GARA provides three different resource managers:

- A differentiated services network resource manager to provide quality of service over a network.
- A CPU resource manager that uses the Dynamic Soft Real-Time (DSRT) scheduler[1] for controlling scheduling for a processes.
- A DPSS resource manager that allows exclusive access to a DPSS server.

Of these three resource managers, you are mostly likely to use the differentiated services and DSRT resource managers.

To use GARA, any resource manager that you need must be running. The resource managers are daemons that are executed once and left to run.

Installing GARA

This section describes how to obtain, compile, and configure GARA.

Obtaining GARA

Currently, GARA is available by request only. If you would like to receive the latest version of GARA, please send mail to roy@mcs.anl.gov. Make sure to include ÒGARAÓ in the subject line of your mail. We will be happy to provide you with a tarball containing the latest release of GARA.

The latest information about GARA is available online from: http://www.mcs.anl.gov/qos/.

Before You Compile GARA

Before you can compile GARA, you must have Globus properly installed. More information about Globus can be found at http://www.globus.org. Note that to install Globus, you will have to first install SSLeay, a set of libraries providing security for Globus.

As part of your Globus installation, you will need to have:

- The ability to run a Globus gatekeeper. It can either be fully deployed so that you can access it through inetd, or you can run it from the command-line. If you wish to have it accessible through inetd, you will need to have requested a certificate for the gatekeeper.

[1] DSRT was produced independently by the MONET group at the University of Illinois at Urbana-Champagne. Although DSRT has been included in the GARA distribution, you can obtain the latest version and information about DSRT from: http://cairo.cs.uiuc.edu.

- A certificate that allows you to be authenticated and authorized through the gatekeeper. To learn more about requesting certificates for Globus, please refer to http://www.globus.org/security/.
- A proper deployment so that you can compile GARA using the Globus include files and libraries.

If you will be using the differentiated services resource manager for network reservations, you will also need to have Tcl/Tk installed. The diffserv implementation of GARA requires that the Tcl/Tk toolkit with Expect is installed on your system. You can learn more about Tcl/Tk and download a recent version from http://expect.nist.gov. Expect is used for a single short script that configures the routers with telnet. More information about this script is available below, under *Configuring the setup_flow Script*.

Compiling GARA

Compiling GARA is quite straightforward. Assume that you have obtained GARA and placed it into your home directory: `~/gara/`. Within this directory is a script called `build`. It looks olike this:

```
#!/bin/sh
CC=cc
CXX=CC
CFLAGS=Ò-g -vÓ
GLOBUS_DIR=/home/roy/globus/development/sparc-sun-
solaris2.7_pthreads_standard_debug
SSL_LIBRARY_DIR=/soft/pub/packages/SSLeay-0.9.0/lib

export CC CXX CFLAGS GLOBUS_DIR SSL_LIBRARY_DIR

autoconf
configure
make clean
make depend
make all
#
# For portability reasons you might not want to do a make withdsrt..
# in case: just comment out this line.
make withdsrt
```

To compile GARA, you need to modify the five variables at the top of this script:

- *CC*: This indicates what C compiler you are using.
- *CXX*: This indicates what C++ compiler you are using.
- *CFLAGS*: This indicates what flags should be passed to your compiler. In the example above, which was on a Solaris installation, the –g option indicates that debugging information should be included, and the –v option indicates that warnings should be printed out during compilation. Note that for gcc, –v indicates that version information should be printed, whereas —Wall turns on printing of warnings.
- *GLOBUS_DIR*: The location where the Globus include files and libraries can be found.

- *SSL_LIBRARY_DIR*: The location where the SSL include files and libraries can be found.

If you will not be using DSRT, you can comment out the final line (make withdsrt). In this case, the C++ compiler will not be used, just the C compiler.

Once you have changed these variables to be appropriate for your system, you can simply type ÒbuildÓ to execute this script and build GARA.

Assuming that build executes with no errors, you will find the various pieces of GARA located within the GARA directories: currently there is no Òmake installÓ to put everything into a uniform location. Assuming that you have GARA in ~/gara/, you will find items in the following locations.

Resource Managers
~/gara/resource_manager/programs/diffserv_manager
~/gara/resource_manager/programs/dsrt_manager
~/gara/resource_manager/programs/dpss_manager

GARA Gatekeeper Service
~/gara/gara/programs/globus_gara_gatekeeper_service

Test programs
~/gara/resource_manager/tests/diffserv_manager_test
~/gara/resource_manager/tests/dsrt_manager_test
~/gara/resource_manager/tests/dpss_manager_test
~/gara/lram/lram_diffserv_test
~/gara/lram/lram_dsrt_test
~/gara/gara/tests/gara_test

Configuring Cisco Routers (Network Only)

We are currently using CiscoÕs Modular QoS command line interface (MQC) for marking and policing traffic on the edge routers. On the ingress side of the first-hop router, GARA uses an existing, attached service-policy Òset-precedenceÓ to mark packets of a specific flow with the differentiated services codepoint (DSCP) 46. Policing is implemented with a token bucket mechanism. Every premium flow has its own token bucket with a filling rate equivalent to the reserved rate and with a depth that is configured by GARA. Note that we configure the policing so that if your applications send more than their reservation, the excess traffic will be dropped. This is similar to the Differentiated Services Working GroupÕs Expedited Forwarding PHB, and we see it as important for ensuring that reserved traffic doesnÕt overwhelm other traffic. You can change this behavior by editing the setup_flow script. (See *Configuring the setup_flow Script*, below.)

For policing, we are currently using a token bucket depth that depends on the reservationÕs bandwidth. For TCP the depth is (bandwidthÄ40), which relates to an estimated maximum roundtrip-time of 25ms. For UDP we use a quarter of this value. There is no excess burst used.

To use the current CLI, our resource manager assumes that each input interface has an attached service-policy named Òset-precedenceÓ The resource manager will modify this service-policy automatically.

Configuration steps:

```
#conf term
#policy-map set-precedence
#exit
#int <name-of-your-input-interface>
#service-policy input set-precedence
#exit
#exit
```

In addition to GARAÕs run-time configuration of the edge routers, you will need to pre-configure the interior router interfaces to treat marked packets differently. Cisco provides several mechanisms for this. Weighted Random Early Detection (WRED), Weighted Fair Queuing (WFQ) and Priority Queuing (PQ) are the possible methods. In our testbed, we currently use WFQ and PQ. Note that this configuration has to be done only once, before any reservations have been.

We recommend the following configuration steps on the interior interfaces for WFQ and PQ.

```
# conf term
# class-map dscp
# match ip dscp 46
# exit
# class-map priority-qos
# match qos-group 99
# exit
# policy-map qos-control
# class dscp
# bandwidth <bandwidth_under_congestion_in_kbits_per_second>
# shape average <actual_reserved_bandwidth_plus_layer_2_overhead>
# queue 500
# exit
# class priority-qos
# priority <optional bandwidth for low-latency>
# exit
# class class-default
# queue 500
# exit
# exit
# int <name-of-interior-interface>
# service-policy output qos-control
# exit
# exit
```

Note: The `bandwidth_under_congestion_in_kbits_per_second` parameter contains the full ATM overhead. Similarly, the bandwidth specified by the shape command also includes the full ATM overhead. However, the latter is configured by the setup_flow script. The idea of the shaping is to prevent premium traffic from exceeding the assigned bandwidth in the core

network. The WFQ bandwidth is not updated to reflect the actual amount of reserved bandwidth because the software is not able to handle this operation with an attached service-policy, therefore shaping takes care of this.

Configuring the setup_flow Script (Network Only)

Configuring the setup_flow script is perhaps the most problematic part of configuring GARA. The setup_flow script uses Expect with Tcl/Tk to telnet to a Cisco router and configure it. There are two distinct elements that you may need to configure: The setup_flow script and the setup_flow.cfg file.

The setup_flow script

The setup_flow script, as distributed, assumes that you are using Cisco's TACACS+, which requires a username and password to log in. Furthermore, it assumes that once you log in, you do not need to enter an "enable" command with a password to gain access to the configuration commands that are needed to configure the router.

If you are not using TACACS+, or you need to use an "enable" command, you will need to modify the setup_flow script. Don't worry! It's fairly straightforward.

Around line 196 of the setup_flow script, you'll see some commands that look like this:

```
spawn telnet $router
expect_after default { close; continue }

expect  "name: "      { send "$username\r" }
expect  "word: "      { send "$password\r" }

expect "#"            { send "configure terminal\r" }
```

When the script is waiting for the router to send "Password", the script uses the "expect "name:*"" command. In response, it sends the username, which is a variable configured previously. Afterwards, it waits for the prompt, and sends "configure terminal"

If you router expects a different interaction, or provides different prompts, you can change the interaction here. For example, if the router only prompts for a password, then expects you to use "enable", you might use:

```
spawn telnet $router
expect_after default { close; continue }

expect  "word: "      { send "$password\r" }
expect  "> "          { send "enable\r" }
expect  "word: "      { send "$password\r" }
expect  "#"           { send "configure terminal\r" }
```

The setup_flow.cfg file

The setup_flow script is configured through the setup_flow.cfg file. Currently, itÕs a bit clumsy to configure, but this will be changing in future versions of GARA.

Within the configuration script, you configure, for each router:

- The login ID and password.
- The IP address used to telnet to the router.
- The name of each ingress interface that can be configured dynamically.
- For each interface, the computers that are nearest to that interface. This is so the setup_flow script can decide which interface is the appropriate one to configure to provide the quality of service for a particular computer.

You will specify the routers as 1_ROUTER, 2_ROUTER, 3_ROUTER, and so forth. For router 1, you will specify the interfaces as 1_INTERFACE1, 1_INTERFACE2, and so forth. Router 2 would be 2_INTERFACE1, 2_INTERFACE2, and so forth.

Here is an example:

```
######################################################################
#
# First specify the ingress router, which has two interfaces. Each
# interface has a single computer attached to it.
#
1_ROUTER          ingressrouter.mcs.anl.gov
1_USERNAME        gara_user
1_PASSWORD        some_clever_password
1_INTERFACE1      atm4/0/0 140.221.48.100
1_INTERFACE2      atm5/0/0 140.221.49.100
#
# The next router
#
2_ROUTER          egressrouter.mcs.anl.gov
2_USERNAME        username
2_PASSWORD        password
2_INTERFACE1      atm4/0/0 140.221.38.100
2_INTERFACE2      atm5/0/0 140.221.39.100
#
######################################################################
```

If you needed to modify the setup_flow script, as described above, your changes may impact how you create this configuration file. For example, you may choose to put the ÒenableÓ password into this configuration file.

Configuring the Resource Manager

You can configure the resource manager with a configuration file, or with command-line options when you run the resource manager. Command-line options always override options in the configuration file.

Here is a sample configuration file for the differentiated services resource manager. It is located in a file named "`diffserv_manager.conf`" in the same directory as the `diffserv_manager` executable.

```
# A Configuration file for the Diffserv Manager
Port 5692
Quantity 10000
ClearSlotTable true
Verbose true

# First we must specify the number of ingress/egress router
# The current version supports up to 5 edge routers
NoOfRouters 2
# Then the addresses served by them
# The resolution of router address end-point address is
# done in setup_flow.cfg
IPAddressesServed[1] 140.221.48.162,140.221.48.146
IPAddressesServed[2] 140.221.48.98,140.221.48.114
```

"Port" is the port used for TCP communication with the resource manager.
"Quantity" is the amount of bandwidth that the resource manager will allow to be given out. It is in kilobits per second.
"Verbose true" indicates that the resource manager will print information about what it happening to the screen.
"NoOfRouters" is the number of routers that can be configured.
"IPAddressesServed[n]" are the machines that are controlled by the routers. (This is only needed for the diffserv_manager, not the dsrt_manager or the dpss_manager.) You will notice that this information is duplicated in setup_flow.cfg. A future version of GARA will unify these configuration files to avoid that. Note that if an address is not in the configuration file, the resource manager will not allow a network reservation to be made.

For more details about configuring the resource manager, see the *Reference* below.

Using the DSRT Manager

For the DSRT manager to work correctly, you will need to have run the DSRT program `CpuSvc` as well. For more information about this, see ~/gara/dsrt.

Configuring the Gatekeeper

Globus has a file named "`grid-services.conf`". This file informs the gatekeeper what programs to run in response to different requests. For example, when you submit a job to the "fork" service, the grid- services.conf file will have information about where the fork job-manager process is.

For GARA, you need to add a line describing where the GARA gatekeeper service is. If you have installed GARA in `/homes/roy/gara/`, then the gatekeeper service will be in `~/gara/gara/programs/globus_gatekeeper_gara_service`, and you will add a line to the `grid-services.conf` file like this:

```
gara-service stderr_log -
/homes/roy/gara/gara/programs/globus_gatekeeper_gara_service gara_service
```

(Note that this is all one on line, although it has been wrapped here.)

This indicates that the gatekeeper should respond to requests for the `gara-service` by running the `globus_gatekeeper_gara_service` program with a single argument Ò`gara-service`Ó

Testing GARA

This section describes some basic tests to make sure that GARA is functioning properly. In all of the following descriptions, it is assumed that you have installed GARA in ~/gara/. If you have installed it elsewhere, you will need to adjust the paths of the programs that are used.

Testing the setup_flow script (Network Only)

You can run the setup_flow script from the command line to make sure that it runs correctly with your Cisco routers. In normal usage, the diffserv_manager will always run the script for you. Setup_flow has a slew of command-line parameters:

```
setup_flow setup/teardown setup_flow.cfg source-ip-address source-port
  destination-ip destination-port bits-per-second burst-size
  burst-size-exceed acl-to-use protocol priority-queueing
```

Here are what the parameters mean:

- *setup/teardown:* If you use Òsetup,Ó the reservation will be configured on the router. If you use Òteardown,Ó the configuration that was previously made will be removed.
- *source-ip-address:* The numeric dotted IP address of the sender.
- *source-port:* The port of the sender
- *destination*-ip*:* The numeric dotted IP address of the receiver.
- *destination-port:* The port of the receiver.
- *bits-per-second:* How fast data can be sent, in bits per second.
- *burst-size:* The maximum allowable burst, in bits per second.
- *burst-size-exceed:* How long the burst can be, inÉ?
- *acl-to-use:* A number between 100 and 199, inclusive. Each reservation has a unique ACL.
- *protocol:* Either Òtcp,Ó or Òudp.Ó
- *priority-queuing:* Either 1, to indicate the priority-queuing should be used, or 0, to indicate that it should not be used.

You should both setup and teardown a reservation using the setup_flow script. If there are any errors in the interaction with the router, this is the time to discover and repair them.

Testing the Resource Manager

Run the manager that you would like to test:

- diffserv_manager, for the differentiated services implementation
- dsrt_manager, for CPU reservations.
- dpss_manager, for DPSS reservations.

In ~/gara/resource_manager/tests, you will find corresponding test programs:

- diffserv_manager_test, to test the differentiated services implementation
- dsrt_manager_test, to test the CPU reservations.
- dpss_manager_test, to test the DPSS reservations.

Each one takes a simple set of parameters:

`diffserv_manager_test <ip-address-1> <ip-address-2>`
: The ip-addresses are the ones of the computers involved in the reservation, and they should be in the diffserv_manager.conf file and the setup_flow.cfg file.

`dsrt_manager_test`
: This should take a process ID as a parameter, but it doesn't.

`dpss_manager_test`
: This should take a process ID as a parameter, but it doesn't.

Testing the LRAM Layer

First make sure that the appropriate resource manager is running. You will find two LRAM test programs in ~/gara/lram:

- lram_diffserv_test: to test the differentiated services implementation
- lram_dpss_test: to test the CPU reservations.

They are run exactly as the resource manager tests above, but they use the LRAM API (which is also used in the gara gatekeeper service) to communicate with the resource manager.

Testing the GARA API Layer

Currently, there is only a test for the GARA API that makes differentiated services reservations.

You will find a program named `gara_test` in ~/gara/gara/tests. This is a full-blown test of the GARA API, and therefore you need to run `grid-proxy-init` before you execute `gara_test`, or you will not be authenticated properly. The basic test requires three parameters: the gatekeeper contact string, and two valid IP addresses (addresses that are in the diffserv_manager.conf.). An example use of `gara_test` might look like:

```
./gara_test -c "dslnet2.mcs.anl.gov:754:/C=US/O=Globus/O=Argonne
National Laboratory/OU=Mathematics and Computer Science
Division/CN=dslnet2.mcs.anl.gov" –1 140.221.48.162 –2 140.221.48.98
```

Note that often the gatekeeper contact string will need to be contained in quotes because many contact strings contain spaces.

Reference

Configuration Options for the Resource Manager

Each resource manager can be configured either on the command line or in a configuration file. Options specified on the command line always override options specified in a configuration file.

Here are the command-line parameters that you may use:

-p <number>	The port number to use for communication. Defaults: 5692 for the diffserv_manager 5693 for the dsrt_manager 5694 for the dpss_manager
-q <number>	The quantity that can be reserved. For the dsrt_manager, this should be 0-1. For the diffserv_manager, this is in kbps.
-f <name>	The name of the slot table where reservations are stored. Defaults: diffserv_manager.slot_table for the diffserv_manager dsrt_manager.slot_table for the dsrt_manager dpss_manager.slot_table for the dpss_manager
-c	Clear slot table. If you donÕclear it, reservations that were made during a previous invocation of the resource manager will be kept.
-x	DonÕreally bind reservations. This will do everything except provide the QoS to the reservations: routers wonÕbe configured, and DSRT wonÕbe used.
-v	Give verbose messages.
-m <method>	Publication method. This can be one of: web: Save in an html file file: Save in a human-readable file. mds: Save in the MDS.
-h	help
-l	location of logging file (defaults to name.log, where name is the name of the resource manager. For exampl,e Òdiffserv_manager.logÓ

Configuration files are named simply:

Resource Manager	Configuration File
diffserv_manager	diffserv_manager.conf
dsrt_manager	dsrt_manager.conf
dpss_manager	dpss_manager.conf

Here are the commands you may use in configuration files:

Port <number>	The port that the resource manager listens to. See above.
Quantity <amount>	The quantity that can be reserved. See above.
SlotTableFilename <name>	The file name that the is used to store reservations. See above.
ClearSlotTable <true | false>	Clear the slot table, or not, depending on Òtrue or ÒfalseÓ
DontReallyBindReservations <true | false>	DonÕt actually provide QoS. See above.
Verbose <true | false>	Turn on verbose mode. See above.
Publication Method	See —m above.
LDAPDistinguishedName	Where to publish the MDS information.
MDSPublishPassword	The password to use to access the MDS
WEBPublishFile	The file to publish the HTML file into
FILEPublishFile	The file to publish the human-readable reservation information into.
LoggingFileName	The file to save the running log of what happens in the resource manager. See the —l option above.

Note that there are a few additional commands used in the diffserv_manager.conf file, and they are essential:

NoOfRouters	How many routers are configured
IPAddressesServed[n]	The list of IP addresses (numeric dotted form) attached to a router. These are the addresses that can receive QoS.
SmallestACL	The smallest ACL that can be used by the router. Defaults to 100.
LargestACL	The largest ACL that can be used by the router. Defaults to 199.

Contact Information

For more information about GARA, you can refer to the GARA web page: http://www.mcs.an.gov/qos/.

For technical information about GARA, as well as information about collaborations and plans for future work, you should contact:

Ian Foster(foster@mcs.anl.gov)
Alain Roy (roy@mcs.anl.gov)
Volker Sander(sander@mcs.anl.gov)